International Journal of Data Science and Analytics (2022) 13:33–46
https://doi.org/10.1007/s41060-021-00288-8

REGULAR PAPER

Graph sparsification with graph convolutional networks

Jiayu Li[1] · Tianyun Zhang[2] · Hao Tian[1] · Shengmin Jin[1] · Makan Fardad[1] · Reza Zafarani[1]

Received: 2 November 2020 / Accepted: 6 September 2021 / Published online: 13 October 2021

Abstract
Graphs are ubiquitous across the globe and within science and engineering. Some powerful classifiers are proposed to classify nodes in graphs, such as Graph Convolutional Networks (GCNs). However, as graphs are growing in size, *node classification* on large graphs can be space and time consuming due to using whole graphs. Hence, some questions are raised, particularly, whether one can prune some of the edges of a graph while maintaining prediction performance for node classification, or train classifiers on specific subgraphs instead of a whole graph with limited performance loss in node classification. To address these questions, we propose *Sparsified Graph Convolutional Network* (SGCN), a neural network graph sparsifier that sparsifies a graph by pruning some edges. We formulate sparsification as an optimization problem and solve it by an Alternating Direction Method of Multipliers (ADMM). The experiment illustrates that SGCN can identify highly effective subgraphs for node classification in GCN compared to other sparsifiers such as Random Pruning, Spectral Sparsifier and DropEdge. We also show that sparsified graphs provided by SGCN can be inputs to GCN, which leads to better or comparable node classification performance with that of original graphs in GCN, DeepWalk, GraphSAGE, and GAT. We provide insights on why SGCN performs well by analyzing its performance from the view of a low-pass filter.

Keywords Graph sparsification · Node classification · Graph convolutional network

1 Introduction

Graphs have become universal and are growing in scale in many domains, especially on the Internet and social media. Addressing graph-based problems with various objectives has been the subject of many recent studies. Examples include studies on link prediction [20] and graph clustering [26], or on node classification [2], which is the particular focus of this study.

In node classification, one aims to classify nodes in a network by relying on node attributes and the network structure. There are two main categories of node classification methods: (1) methods that directly use node attributes and structural information as features and use classifiers (e.g., decision trees) to classify nodes, and (2) random walk-based methods, which classify nodes by determining the probability p that a random walk starting from node $v_i \in V$ with label c will end at a node with the same label c. The performance of random walk-based methods implicitly relies on graph structural properties, e.g., degrees, neighborhoods, and reachabilities.

In recent studies, neural network classifiers [35] are widely used for both types of methods due to their performance and flexibility. A well-established example is the Graph Convolutional Network (GCN) [17], a semi-supervised model that uses the *whole* adjacency matrix as a *filter* in each neural network layer. However, there is a major difficulty faced by methods that directly use the whole graph to extract structural information: the size of the graph. Unlike node attributes, as a graph with n nodes grows, the size of its adjacency matrix

✉ Jiayu Li
jli221@data.syr.edu

Tianyun Zhang
t.zhang85@csuohio.edu

Hao Tian
haotian@data.syr.edu

Shengmin Jin
shengmin@data.syr.edu

Makan Fardad
makan@syr.edu

Reza Zafarani
reza@data.syr.edu

[1] Syracuse University, Syracuse, USA

[2] Cleveland State University, Cleveland, USA

increases at a n^2 rate, which introduces an unavoidable space and computational cost to classifiers. One engineering solution is to store the adjacency matrix as a sparse matrix (i.e., save non-zeros); however, the process is still extremely slow and requires massive storage when the graph is dense or large. Meanwhile, more edges in the graph can introduce over-fitting in node classification.

The present work: sparsified graph convolutional network (SGCN). Inspired by the challenge in node classification, we explore whether we can keep the skeleton of a graph (i.e., a subgraph), such that the skeleton can be used as the input to node classification instead of the whole graph while maintaining the state-of-art performance. We propose *Sparsified Graph Convolutional Network* (SGCN), a neural network graph sparsifier to prune the input graph to GCN without losing much accuracy in node classification. We formulate graph sparsification as an optimization problem, which we efficiently solve via the Alternating Direction Method of Multipliers (ADMM) [3]. We also introduce a new gradient update method for the pruning process of the adjacency matrices, ensuring updates to the matrices are consistent within SGCN layers.

To evaluate SGCN, we compare its performance with other classical graph sparsifiers on multiple real-world graphs. We demonstrate that within a range of pruning ratios, SGCN provides better-sparsified graphs for GCN when compared to other graph sparsifiers. We also show that node classification performance using these sparsified graphs in GCN can be better or comparable to when (much larger) original graphs are used in GCN, DeepWalk [22], GraphSAGE [14] and GAT [33]. As an extended version of our original paper on SGCN [18], we conduct experiments to demonstrate that as GCN becomes deeper, the performance of GCN combined with SGCN is more stable on node classification than that of GCN. Moreover, by analyzing the performance of the low-pass filter produced by SGCN, we theoretically explain the effect of SGCN on GCN. In sum, our contributions can be summarized as:

1. We propose Sparsified Graph Convolutional Network (SGCN), the first neural network graph sparsifier that sparsifies graphs to enhance node classification;
2. We design a gradient update method that ensures adjacency matrices in the two SGCN layers are updated consistently;
3. We demonstrate that the sparsified graphs obtained by SGCN perform better in GCN for node classification that those provided by other graph sparsifiers;
4. We show that sparsified graphs obtained from SGCN with various pruning ratios, if used as inputs to GCN, lead to classification performances similar to that of GCN, DeepWalk, GraphSAGE, DropEdge-GCN, and GAT using the (much larger) whole graphs;
5. We apply experiments to demonstrate that using subgraphs obtained by SGCN in GCN yields more stable results in node classification when the depth of GCN increases; and
6. We theoretically analyze and explain the effect of SGCN on GCN from the view of a low-pass filter in Sect. 6.

The paper is organized as follows. We review related work in Sect. 2. We provide the SGCN problem definition in Sect. 3. Section 4 details the problem formulation, solution, and time complexity of SGCN. We conduct experiments in Sect. 5. The performance analysis of SGCN from the view of a low-pass filter is provided in Sect. 6. We conclude in Sect. 7.

2 Related work

2.1 Graph neural networks

Inspired by the major success of convolutional neural networks in computer vision research, new convolutional methods have emerged for solving graph-based problems. There are two main types of graph convolutional networks: *spectral-based* methods and *spatial-based* methods.

Spectral-based convolutional networks often rely on graph signal processing and are mostly based on normalized graph Laplacian. GCNs [9,17] process the whole graph as input to the neural network. They face various challenges for efficient processing of large-scale graphs. For addressing these challenges, different variants of GCNs are proposed. Cluster-GCN [7] is based on graph clustering and proposes a stochastic multi-clustering framework which can improve the efficiency of GCNs while handling large-scale graphs. SGC [34], an efficient GCN, reduces the excess complexity of GCNs by repeatedly removing the nonlinearities between GCN layers and collapsing the resulting function into a single linear transformation. FastGCN [5] solves the recursive neighborhood expansion across layers, which causes time and memory challenges for training with large, dense graphs. The method interprets graph convolutions as integral transforms of embedding functions under probability measures. Such an interpretation allows for the use of Monte Carlo approaches to consistently estimate the integrals, which in turn leads to a batched training scheme. Other examples include the work of Bhagat et al. [21], which aims to represent a graph by extracting its locally connected components. Another is DUIF, proposed by Geng et al. [12], which uses a hierarchical softmax for forward propagation to maximize modularity. One main drawback of spectral-based methods is the need to perform matrix multiplication on the adjacency matrix, which is costly for large graphs.

Spatial-based methods focus on aggregating the neighborhood for each node. These methods can be grouped into (1) recurrent-based and (2) composition-based methods. Recurrent-based methods update latest node representation using that of their neighbors until convergence [8,25]. Composition-based methods update the nodes' representations by stacking multiple graph convolution layers. For example, Gilmer et al. [13] develop a message passing neural network to embed any existing GCN model into a message passing (the influence of neighbors) and readout pattern. GraphSAGE [14] operates by sampling a fixed-size neighborhood of each node and performing a specific aggregator over it, which yields impressive performance across several large-scale inductive benchmarks. GAT [33] applies attention mechanism and enables graph nodes having different degrees by assigning arbitrary weights to the neighbors, without requiring any costly matrix operations or dependence on knowing the graph structure. Spatial-based methods are often more flexible and easier to apply to large networks.

2.2 Graph sparsification

For graph sparsification, previous studies have distinct objectives from that of ours. Generally speaking, most graph properties of a dense graph can be approximated from its [sparsified] sparse graph. *Cut sparsifiers* [1,11,16] ensure the total weight of cuts in the sparsified graph approximates that of cuts in the original graph within some bounded distance. Spectral sparsifiers [30,31] ensure sparsified graphs preserve spectral properties of the graph Laplacian. There are various applications for graph sparsification. Some examples include, the work of Serrano et al. [28], which aims to identity the *backbone of a network* that preserves structural and hierarchical information in the original graph; the study by Satuluri et al. [24], which applies local sparsification to preprocess a graph for clustering; the study by Lindner et al. [19], which proposes a local degree sparsifier to preserve nodes surrounding local hub nodes by weighing edges linking to higher degree nodes more, which aims to minimize divergence of stationary distribution of a random walk while sparsifying the graph. Recent research has combined sparsifier with some neural networks as backbone to improve the performance of those neural networks. For example, NeuralSparse [37] proposes edge pruning and utilizes a DNN to parameterize the sparsification process. Similarly, DropEdge [23] considers removing a certain number of edges randomly from an input graph at each training epoch.

These studies are similar, but with different objectives from that of ours. Instead of preserving graph properties or randomly removing edges, our proposed sparsifier SGCN [18] is the first GCN-based sparsification by formulating and solving it as an optimization problem. Finally, the space cost is reduced due to sparsification, while node classification performance of GCN is maintained, or improved.

3 Problem definition

Consider an undirected graph $G = (V, E)$, its nodes $V = \{v_1, \ldots, v_n\}$, and its edges $E = \{e_1, \ldots, e_m\}$. Let $n = |V|$ denote the number of nodes and $m = |E|$ denote the number of edges. Given adjacency matrix A of G and features for each node $v : X(v) = [x_1, \ldots, x_k]$, the forward model (i.e., output) of a two-layered graph convolutional network (GCN), as formulated by Kipf and Welling [17], is

$$Z(\hat{A}, W) = \text{softmax}(\hat{A}\ \text{ReLU}(\hat{A}XW^{(0)})W^{(1)}), \tag{1}$$

where $\hat{A} = \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$, $\widetilde{D} = \text{diag}(\sum_j \widetilde{A}_{ij})$, $\widetilde{A} = A + I_N$, X is the matrix of node feature vectors $X(v)$, and $W^{(0)}$ and $W^{(1)}$ are the weights in the first and second layer, respectively. Functions $\text{softmax}(x_i) = \exp(x_i)/\sum_i \exp(x_i)$ and $\text{ReLU}(\cdot) = \max(0, \cdot)$ both perform entry-wise operations on their arguments. Graph sparsification aims to reduce the number of edges $|E|$ in the original graph G to $|E_s|$ in a subgraph G_s, i.e., $|E_s| < |E|$, such that the induced subgraph G_s, when used as input to GCN, results in similar classification performance to that of the original graph G. In pruning, adjacency A is pruned to $A_p = A - B \odot A$, where B is a matrix and $\odot$ is Hadamard product. Thus, the new $\widetilde{A}$ is $\widetilde{A} = A_p + I_N$ and $\hat{A}$ can be an updated filter related to A_p. We explore how the level of graph sparsification in filters affects SGCN performance.

4 SGCN: sparsified graph convolutional networks

We first illustrate the problem formulation and solution, followed by SGCN algorithm, a new gradient update method, and the SGCN time complexity analysis.

4.1 Problem formulation and solution

4.1.1 Problem formulation

The output of graph convolutional networks in Eq. (1) is a function of $\hat{A}$ and W, but as $\hat{A}$ can be written as a function of A, the output can be written as $Z(A, W)$. For semi-supervised multiclass classification, loss function of the neural networks is the cross-entropy error over labeled examples:

$$f(A, W) = -\sum_{l \in \mathcal{Y}_L} \sum_{f} Y_{lf} \ln(Z_{lf}) \tag{2}$$

where $\mathcal{Y}_L$ is the set of node indices that have labels, Y_{lf} is a matrix of labels and Z_{lf} is the output of the GCN forward model.

Our aim is to achieve a sparse graph, with weight matrices being fixed in SGCN. In the following, we will use $f(A)$ to present the loss function and our problem is defined by:

$$\begin{aligned} &\underset{A}{\text{minimize}} \quad f(A) \\ &\text{subject to} \quad \|A\|_0 \leq l \end{aligned} \tag{3}$$

For Eq. (3), we define an indicator function to replace constraint:

$$g(\Lambda) = \begin{cases} 0 & \text{if } \|\Lambda\|_0 \leq l, \\ +\infty & \text{otherwise,} \end{cases}$$

Therefore, formulation (3) can be rewritten as

$$\underset{A}{\text{minimize}} \quad f(A) + g(A). \tag{4}$$

4.1.2 Solution

In Eq. (4), the first term $f(\cdot)$ is the differentiable loss function of the GCN, while the second term $g(\cdot)$ is the non-differentiable indicator function; hence, problem (4) cannot be solved directly by gradient descent. To deal with this issue, we propose to use Alternating Direction Method of Multipliers (ADMM) to rewrite problem (4). ADMM is a powerful method for solving convex optimization problems [3]. Recent studies [15,32] have demonstrated that ADMM also works well for some nonconvex problems.

The general form of a problem solvable by ADMM is

$$\begin{aligned} &\underset{\alpha,\, \beta}{\text{minimize}} \quad f(\alpha) + g(\beta), \\ &\text{subject to} \quad P\alpha + Q\beta = r. \end{aligned}$$

The problem can be decomposed to two subproblems via augmented Lagrangian. One subproblem contains $f(\alpha)$ and a quadratic term of α; the other contains $g(\beta)$ and a quadratic term of β. Since the quadratic term is convex and differentiable, the two subproblems can often be efficiently solved.

Hence, we rewrite problem (4) as

$$\begin{aligned} &\underset{A}{\text{minimize}} \quad f(A) + g(V), \\ &\text{subject to} \quad A = V. \end{aligned} \tag{5}$$

The augmented Lagrangian [3] of problem (5) is given by

$$L_\rho(A, V, \Lambda) = f(A) + g(V) + \text{tr}[\Lambda^T (A - V)] + \frac{\rho}{2} \|(A - V)\|_F^2,$$

where Λ is the Lagrangian multiplier (i.e., the dual variable) corresponding to constraint $A = V$, the positive scalar ρ is the penalty parameter, $\text{tr}(\cdot)$ is the trace, and $\|\cdot\|_F^2$ is the Frobenius norm.

By defining the scaled dual variable $U = (1/\rho)\Lambda$, the augmented Lagrangian can be equivalently expressed in the scaled form:

$$L_\rho(A, V, U) = f(A) + g(V) + \frac{\rho}{2} \|A - V + U\|_F^2 - \frac{\rho}{2} \|U\|_F^2.$$

When we apply ADMM [3] to this problem, we alternately update the variables according to

$$A^{k+1} := \underset{A}{\arg\min} \quad L_\rho(A, V^k, U^k), \tag{6}$$

$$V^{k+1} := \underset{V}{\arg\min} \quad L_\rho(A^{k+1}, V, U^k), \tag{7}$$

$$U^{k+1} := U^k + A^{k+1} - V^{k+1}, \tag{8}$$

until

$$\|A^{k+1} - V^{k+1}\|_F^2 \leq \epsilon, \quad \|V^{k+1} - V^k\|_F^2 \leq \epsilon. \tag{9}$$

In (6), we solve the first subproblem:

$$\underset{A}{\text{minimize}} \quad f'(A) := f(A) + \frac{\rho}{2} \|A - V^k + U^k\|_F^2 . \tag{10}$$

In the above problem, as the loss function $f(A)$ and the ℓ_2-norm are differentiable, we can use gradient descent to solve it. As $f(A)$ is nonconvex with respect to the variable A, there has been no theoretical guarantee on the convergence when solving problem (10). We present a method to solve (10) in Sect. 4.3.

In (7), we solve the second subproblem , which is

$$\underset{V}{\text{minimize}} \quad g(V) + \frac{\rho}{2} \|A^{k+1} - V + U^k\|_F^2 . \tag{11}$$

As $g(\cdot)$ is the indicator function, problem (11) can be solved analytically [3], where the solution is

$$V^{k+1} = \mathbf{5}_{\mathbf{S}}(A^{k+1} + U^k), \tag{12}$$

where $\mathbf{5}_{\mathbf{S}}(\cdot)$ is the Euclidean projection onto set $\mathbf{S} = \{\Lambda \mid \|\Lambda\|_0 \leq l\}$.

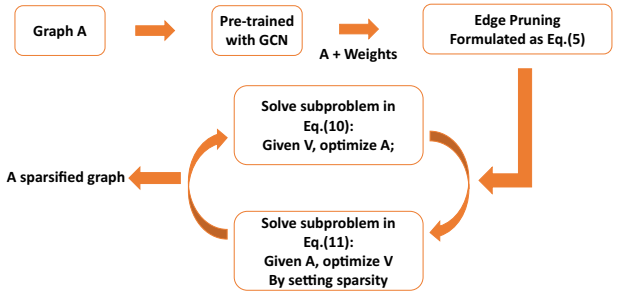


Fig. 1 Framework of SGCN using ADMM. Weights can be pretrained from GCN with an input graph A

Finally, we update the scaled dual variable U according to (8). This is one ADMM iteration. We update the variables iteratively until condition (9) is satisfied, indicating the convergence of ADMM. Figure 1 shows the whole process followed by SGCN. As GCN with two layers can yield better performance, we obtain weights by pretraining a 2-layer GCN. By fixing the weights from the pre-trained GCN, SGCN trains an input graph based on Eq. (3). So, SGCN consists of two graph convolutional layers.

4.2 SGCN algorithm

The ADMM solution discussed facilitates efficient edge pruning in graphs. In the solution, we need to maintain α, the number of non-zero elements. The Euclidean projection in (12) maintains α elements in $\widetilde{A}^{k+1} + U^k$ with the largest magnitude and sets the rest to zero. This is proved to be the optimal and analytical solution to subproblem (11) in edge pruning of graphs. In GCN, filters in the loss function (2) consist of $\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$, where $\widetilde{A} = A + I_N$ and $\widetilde{D} = \sum_j \widetilde{A}_{ij}$. Variable I_N is the identity matrix and $\widetilde{A}$ is a [modified] adjacency matrix. Variables $\widetilde{A}$ and $\widetilde{D}$ in each layer are fixed and non-trainable in the original GCN. To solve graph sparsification in GCN and maintain classification performance, variable $\widetilde{A}$ should be trained and updated iteratively. As variable $\widetilde{D}$ depends on $\widetilde{A}$, $\widetilde{A}$ in the original loss function cannot be directly differentiated. Therefore, we expand the forward model into the following:

$$Z(A) = \text{diag}(\textstyle\sum_j (A+I)_{ij})^{-\frac{1}{2}}(A+I) \times \text{diag}(\textstyle\sum_j (A+I)_{ij})^{-\frac{1}{2}} XW. \quad (13)$$

In Eq. (13), no variable depends on $\widetilde{A}$. However, one cannot still train variables A and W at the same time as the differentiation of A depends on W and vice versa. Hence, we pre-train a model with the original GCN and obtain a pre-trained model with variable W. By fixing variable W in this model, the adjacency matrix A can be regarded as a trainable variable. With ADAptive Moment estimation (ADAM) optimizer, gradients of the variable (adjacency matrix A) can be updated in SGCN. Algorithm 1 provides the pseudo-code SGCN. We use variable A to initialize variable V in each layer using function $Initialize()$ and apply function $Zerolike()$ to V to ensure variable U has the same shape as V with all the zero elements.

Algorithm 1: The SGCN Algorithm

input : Weight matrix W, adjacency matrix A, feature matrix X, ADMM iterations k and pruning ratio $p\%$
output: A pruned adjacency matrix A_p

```
for i ← 1 to the number of layers do
    V_i ← Initialize(A_i);
    U_i ← Zerolike(V_i);
for k ← 0 to ADMM iterations do
    Solve subproblem (10) and update A's in two layers;
    for i ← 1 to the number of layers do
        Update V_i's by performing Euclidean projection (12);
        Update U_i's by performing (8);
Fetch A_1 from the first layer;
Set the smallest p% of nonzero elements in A_1 to zero;
Obtain a pruned adjacency matrix A_p;
```

4.3 Adjacency matrix training

When training adjacency A in Algorithm 1, we should keep the adjacency matrices in the first and second layer consistent. To address this issue, we propose a method to update the gradients of the adjacency matrix when fixing weight matrices W in the two layers during SGCN training. A mask m is defined using the adjacency matrix A. As we use gradient descent, the following equation based on (2) and (13) can be applied to update the trainable variable (adjacency matrix A) at each step to solve problem (10):

$$A_i^{k+1} = A_i^k - \gamma \left(m \odot \frac{\partial f'(A_i^k)}{\partial A_i^k} \right), \quad (14)$$

for $i = 1, \ldots, n$, where γ is the learning rate.

In the process of updating A, we keep the gradient matrices of the adjacency matrix symmetric in two layers and gradients are set to zero when there are no edges between nodes. Also, diagonal elements are zero in the gradient matrix as we only update the adjacency matrix and consider no self-loops at each node. To maintain the adjacency matrices in the two layers identical, we compute average gradients for the same edge in the two adjacency matrices. We assign these average gradients to the corresponding edges in the matrices for updating elements of the adjacency matrices.

Figure 2 illustrates the process of updating gradient of adjacency matrix A in two layers of SGCN. Figure 2a shows the original graph with different edges corresponding to dif-

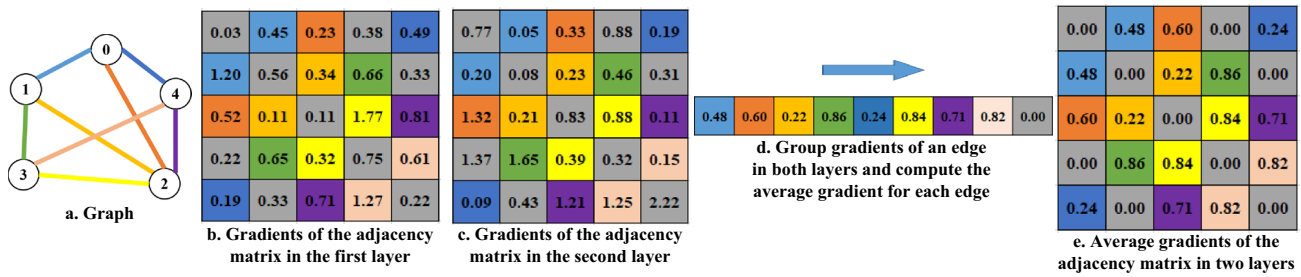


Fig. 2 The process of updating an adjacency matrix in SGCN

ferent colors. Grey color indicates no edges between two nodes. When training SGCN, gradients of each adjacency matrix in two layers are shown in Fig. 2b, c. Gradients are grouped based on their edges (colors) and the average gradient for each edge is calculated in Fig. 2d. We set the average gradient to zero when there are no edges between two nodes. Finally, Fig. 2e displays average gradients are assigned back to their corresponding edges in adjacency matrices. Therefore, when performing gradient descent, the new gradient matrix is used to update each adjacency matrix in the two layers.

4.4 Time complexity analysis

We analyze SGCN training and prediction time complexity. The GCN training time complexity is $\mathcal{O}(L|A_0|F + LNF^2)$, where L is the number of layers, N is the number of nodes, $|A_0|$ is the number of non-zeros in an adjacency matrix, and F is the number of features [7]. Hence, assuming ADMM takes k iterations, the SGCN time complexity is $\mathcal{O}(kL|A_0|F + kLNF^2)$. Compared to SGCN training time complexity, the time to update variables V and U according to (12) and (8) is negligible. In our SGCN, $k = 4$ and $L = 2$. Also, we need only a few iterations to solve subproblem (10), which indicates the training time complexity of SGCN is similar to that of GCN. The time complexity for the forward model in SGCN is $\mathcal{O}(|\Gamma|FC)$, where $|\Gamma|$ is linear in the number of edges and C is the dimension of feature maps. Compared to the SGCN time complexity: $\mathcal{O}(|\epsilon|FC)$, we have $|\Gamma| \leq |\epsilon|$ in prediction.

5 Experiments

There are two natural ways to measure the effectiveness of SGCN.

– First, we compare the node classification performance of GCN using sparsified graphs from SGCN with that using subgraphs obtained from other sparsifiers. Note that stacking many layers in GCN can bring extra computational cost and reduce the accuracy of GCN. Hence, we use 2-layer GCN for node classification, and use the following sparsifiers: *Random Pruning* (RP) sparsifier, *Spectral Sparsifier* (SS) and *DropEdge*. RP removes edges uniformly at random from a graph with some probability. The SS is the state-of-the-art spectral sparsifier [10], which sparsifies graphs in near linear-time. DropEdge [23] randomly removes a certain number of edges from an input graph at each training epoch.[1]
– Second, we compare the node classification performance of the sparsified graphs provided by SGCN compared to that of other node classification techniques that utilize the original graphs. For that, we compare the performance of GCN using sparsified subgraphs provided by SGCN with that of GCN, DeepWalk, GraphSAGE, and GAT using original graphs.

5.1 Experimental setup

5.1.1 Datasets

To evaluate the performance of node classification on sparsified graphs, we conduct our experiments on six attributed graphs. These graphs have been utilized for evaluation in previous studies and are hence used for evaluation. Citeseer, Cora, and Pubmed are from Sen et al. [27] while Terrorists and Terrorist Attacks with high densities are available online.[2] NELL is extracted from a knowledge graph introduced from Carlson et al. [4] with many edges and nodes. We summarize these datasets in Table 1.

5.1.2 Preprocessing

We preprocess the data for existing node classification models [17,36]. We split the data into tenfold for cross-validation. In each training fold, we only select 20 instances for each label as the labeled instances. Other instances remain unlabeled, from which we randomly select 500 instances for

[1] We cannot consider NeuralSparse [37] as the code of NeuralSparse cannot be accessed.

[2] http://linqs.soe.ucsc.edu/data.

Table 1 Dataset statistics

Dataset	Type	Nodes	Edges	Density ($\times 10^{-3}$)
Terrorists	Social network	881	8592	22.16
Terrorist Attacks	Social network	645	3172	15.27
Cora	Citation network	2708	5429	1.48
Citeseer	Citation network	3327	4732	0.86
Pubmed	Citation network	19,717	44,338	0.23
NELL	Knowledge graph	65,755	266,144	0.12

validation set, which is used to train our hyper-parameters. We filter and reorder the adjacency matrices and attribute vectors to ensure they are ordered according to training/testing folds.

5.1.3 Parameter setup

All the experiments implemented with TensorFlow are conducted on a NVIDIA Titan XP GPU (12 GB memory), 4-core Intel Core CPU (3.60 GHz), and 64 GB of RAM. We vary the pruning ratio (p in Algorithm 1) from 10% to 90% in SGCN and RP. When pruning ratio is 0%, the model is the original GCN. We use the default parameters in DropEdge, GCN, DeepWalk, GraphSAGE and GAT. In RP, we set random seeds from 0 to 9. For SGCN, we set ρ to 0.001 and the training learning rate to 0.001. In SS, we use the default suggested parameters for spectral sparsifier [10]. Due to obtaining 10 folds for each dataset, we run SGCN, RP and SS, in each fold of each dataset to obtain sparsified subgraphs and use these subgraphs as inputs for GCN. Codes are released on https://tinyurl.com/594paa4j.

5.2 Results and performance analysis

5.2.1 Comparing sparsifiers

In semi-supervised node classification, Fig. 3 provides the average performance of GCN with sparsified subgraphs obtained from SGCN and other graph sparsifiers. In Fig. 3a, b, d–f, performances are provided in *relative accuracy*, where accuracy is divided by the baseline: accuracy of models from GCN. We cannot consider Spectral Sparsifier (SS) in the first 6 figures because SS does not allow to set a pruning ratio. In Cora dataset, sparsified subgraphs provided by SGCN perform better in GCN than those provided by RP and DropEdge when pruning ratio is less than 20% as shown in Fig. 3a. For CiteSeer dataset, Fig. 3b shows that when the pruning ratio is between 0 and 10%, sparsified subgraph provided by SGCN when used as input to GCN, can yield accurate classification models. When pruning 10% of the edges from the Pubmed dataset in Fig. 3c, we can obtain the best GCN model using the subgraph from SGCN. In Terrorists datasets, applying subgraphs from SGCN as inputs to GCN can easily obtain a higher accuracy, as shown in Fig. 3d. In Fig. 3e on Terrorist Attack dataset, we observe that GCN performance increases as pruning ratio increases in SGCN. Here, also SGCN yields better subgraphs than those provided by RP or DropEdge. Finally, when pruning ratios are between 10% and 20% on NELL dataset, as shown in Fig. 3f, subgraphs from SGCN can provide the best inputs to GCN. Note that DropEdge on NELL dataset has Out-Of-Memory (OOM) issue with our NVIDIA Titan XP GPU (12GB memory) while Rong et al. [23] conduct all experiments on an NVIDIA Tesla $P40$ GPU with 24GB memory. Therefore, SCGN needs less resources when obtaining better subgraphs as inputs to GCN. Figure 3g–l illustrates the performance of GCN using subgraphs from SGCN, RP, SS, and DropEdge. We compare their best performances, and the performance under the same pruning ratio, as for SS we cannot set a pruning ratio. The results show that subgraphs from SGCN perform the best in node classification, and SGCN is more flexible than SS as SGCN allows different pruning ratios. Note that the results on NELL dataset with DropEdge is unavailable due to the limited memory of our device, as shown in Fig. 3l.

5.2.2 Node classification performance

When using sparsified subgraphs provided by SGCN are used as inputs to GCN, we obtain a node-classification model, which we denote as SGCN-GCN. On all datasets, SGCN-GCN either outperforms other methods, or yields comparable performance using much smaller graphs. On Cora and Citeseer dataset (Fig. 4a, b), SGCN-GCN outperforms GraphSAGE and DeepWalk, and has a comparable performance to GAT, GCN and DropEdge-GCN in node classification. On other datasets (Fig. 4c–e), SGCN-GCN outperforms other methods. For NELL dataset shown in Fig. 4f, SGCN-GCN achieves the best performance while GAT and DropEdge-GCN face Out-Of-Memory (OOM) issues. Hence, even though many edges are pruned, subgraphs provided by SGCN when used as inputs to GCN can lead to better or comparable node classification performance over these datasets. The effect of SGCN on GCN is further analyzed mathematically in Sect. 6.

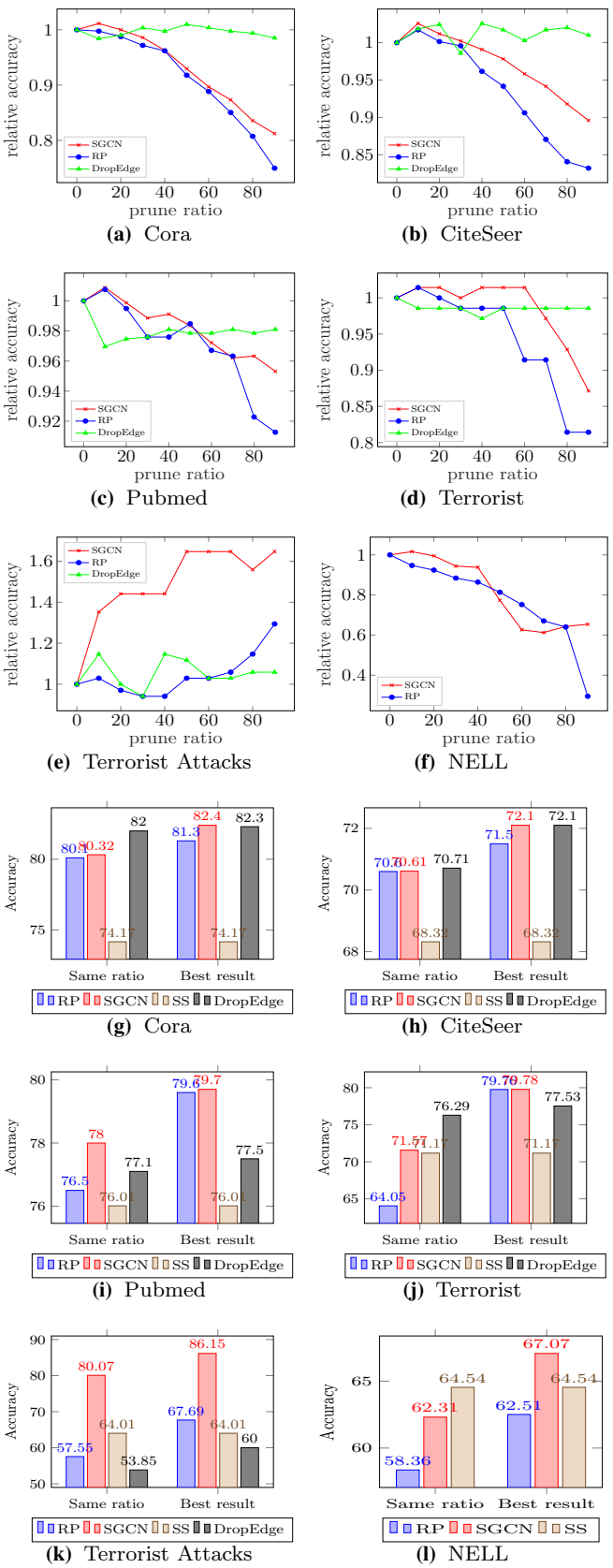
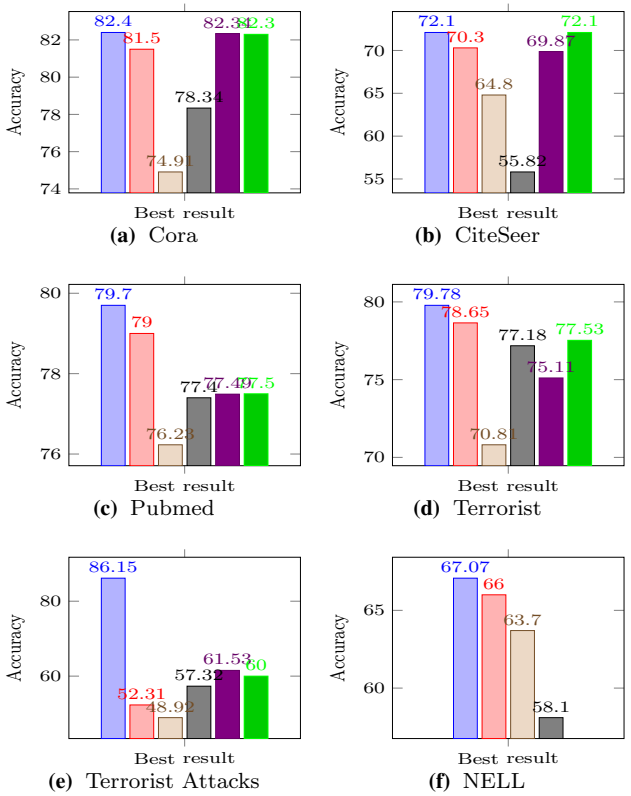


Fig. 3 Performance of GCN using sparsified subgraphs provided by SGCN, RP, SS, and DropEdge sparsifiers

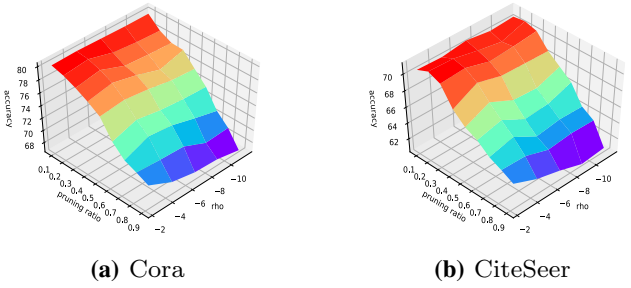


(a) Cora (b) CiteSeer (c) Pubmed (d) Terrorist (e) Terrorist Attacks (f) NELL

Fig. 4 The performance of SGCN-GCN, GCN, GraphSAGE, DeepWalk, GAT and DropEdge-GCN. GCN module is 2 layers among the models

5.2.3 Space and computational cost

Here, we feed the subgraphs from SGCN as inputs to GCN and show the actual space and computational cost. The space cost in a graph is $\mathcal{O}(|V| + |E|)$. As SGCN decreases the number of edges ($|E|$), the space cost is obviously reduced, as shown in Fig. 5a. Figure 5b, c shows average training and prediction times in seconds, which have declining trends when the pruning ratio increases. The reason why we cannot include the results of Pubmed and NELL datasets in the



(a) Cora (b) CiteSeer

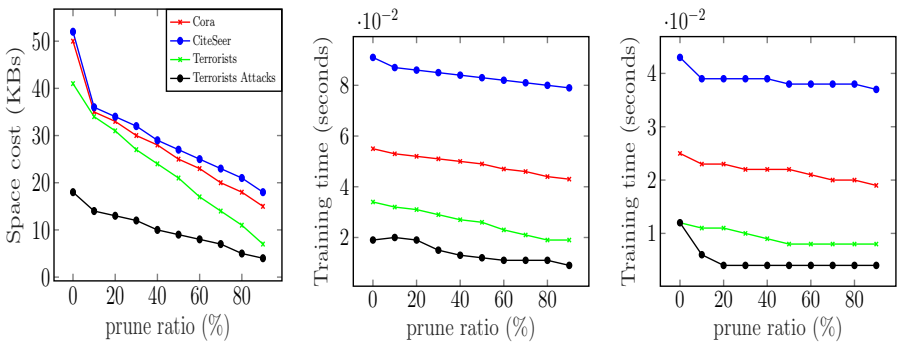
Fig. 6 Parameter analysis. Value of ρ is represented by the logarithm of ρ

figures is that the space and computational reduction from both of the datasets have different scales. On the Pubmed dataset, we observe that when pruning ratios increase from 10 to 90%, the space cost reduces from 433 to 235 KB and the training time decreases from 0.74 to 0.66 s, while we accelerate the prediction from 0.39 to 0.33 s. For NELL dataset, when sparsifying the graph from 1381 to 246 KB, we reduce the training time from 0.24 to 0.13 s and reduce the prediction time from 0.096 to 0.088 s. Hence, the proposed framework significantly reduces both space and computational costs.

5.2.4 Parameter analysis

In the proposed framework, pruning ratio $p\%$ and ρ are used to sparsify graphs. To assess their influence on the method performance, we change the value of $p\%$ from 10% to 90% while set the ρ from 1×10^{-1} to 1×10^{-5} with a step size of 10^{-1}, as shown in Fig. 6. When pruning ratio is less than 20% in Fig. 6a on Cora dataset, we find better GCN models using the sparsified subgraph from SGCN, even though value of ρ is changed from 1×10^{-1} to 1×10^{-5}. In CiteSeer dataset shown in Fig. 6b, pruning ratio has more influence on the performance than the ρ does. Therefore, we conclude that pruning ratio, instead of ρ value, effectively affects the performance of sparsified graphs as inputs to the GCN in node classification.

Fig. 5 Space and computational cost using subgraphs from SGCN as inputs to GCN



(a) Space Cost (b) Average Training Time (c) Prediction Time

Table 2 Performance (%) of model depth (number of layers) on classification. A SGCN-GCN's input is 90% of the original graph (sparsified by the SGCN), while original GCN utilizes the whole graph

Methods	Number of layers							
	2	3	4	5	6	7	8	9
SGCN-GCN(Cora)	81.40	**82.40**	**81.90**	**80.30**	**77.40**	**68.50**	**55.70**	**52.20**
GCN(Cora)	**81.50**	77.60	45.50	67.00	47.60	15.80	15.40	25.60
SGCN-GCN(Citeseer)	**72.10**	**70.40**	**69.00**	**66.80**	**62.70**	**59.20**	**48.60**	**38.20**
GCN(Citeseer)	70.30	66.90	66.50	38.30	16.90	16.90	7.70	24.50
SGCN-GCN(Pubmed)	78.80	**78.40**	**78.90**	**77.40**	73.90	**77.80**	**70.10**	**51.60**
GCN(Pubmed)	**79.00**	77.40	75.50	74.30	**76.10**	18.00	18.00	41.70
SGCN-GCN(Terrorist)	**83.15**	**82.02**	**80.90**	**80.90**	**80.90**	**76.40**	**79.78**	**73.03**
GCN(Terrorist)	78.65	75.28	77.53	76.40	70.79	70.79	60.67	10.11
SGCN-GCN(Terrorist Attacks)	**70.00**	**60.00**	**64.62**	**64.62**	**64.62**	**66.15**	**63.08**	**60.00**
GCN(Terrorist Attacks)	50.77	49.23	52.31	29.23	33.85	30.77	30.77	15.39
SGCN-GCN(NELL)	**67.07**	65.96	62.31	61.70	**63.02**	**62.82**	**57.04**	**58.66**
GCN(NELL)	66.00	**70.11**	**72.23**	**65.35**	44.68	31.51	13.88	13.88

Best results are highlighted in bold

5.2.5 Performance with respect to GCN depth

In Table 2, we explore how subgraphs from SGCN impact the performance of GCN when changing the depth of GCN (number of layers). As shown in the table, we use SGCN to prune 10% of a graph and use the sparsified graph from SGCN as an input to GCN with different number of layers. The results in Table 2 demonstrate that the performance of GCN with subgraphs from SGCN are more stable than that of original GCN as the depth increases.

6 Effect of low-pass filters

Our results indicate that SGCN can sparsify graphs to subgraphs and the subgraphs as inputs to GCN can maintain classification performance. Here, we aim to explain the effect of SGCN on the performance of GCN from the perspective of low-pass spectral filter. Note that SGCN is a neural network sparsifier, and SGCN-GCN is that GCN applies subgraphs from SGCN as inputs.

For a graph Laplacian matrix $\mathbf{L}$, we can obtain $\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^{-1}$, where $\mathbf{U} = [u_0, \ldots, u_N - 1]$ is a matrix of eigenvectors of matrix L, and $\Lambda = diag[\lambda_0, \ldots, \lambda_{n-1}]$ is a matrix of eigenvalues of $\mathbf{L}$. From Graph signal processing (GSP), the *graph Fourier transform* of signal $\mathbf{f}$ is

$$\hat{\mathbf{f}} = \mathbf{U}^{-1}\mathbf{f}, \tag{15}$$

where $\hat{\mathbf{f}}$ is the *graph Fourier coefficients* or *graph spectral coefficients* of signal $\mathbf{f}$. Equivalently, the graph inverse Fourier transform is $\mathbf{f} = \mathbf{U}\hat{\mathbf{f}}$. Therefore, any graph signal can be represented by a linear combination of eigenvectors:

$$\mathbf{f} = \mathbf{U}\hat{\mathbf{f}} = \sum_i \hat{f}_i u_i, \tag{16}$$

where $\mathbf{U}$ can also be a matrix of *Fourier basis vectors* in GSP. From a recent study on Laplacian quadratic form [6], we know that the Fourier basis vectors with small variations are considered as low-frequency components while the vectors with large variations are considered as high-frequency components, which can be used as a measure of smoothness [38]. Shuman and colleagues have demonstrated that eigenvectors associated with smaller eigenvalues have values that vary less rapidly along the edges [29], which indicates that nodes and their neighbors are more likely to have similar features. Therefore, a smooth graph signal $\mathbf{f}$ should mostly consist of low-frequency components with small eigenvalues.

In a recent study *graph convolutional filter* H is defined as $\widetilde{\mathbf{f}} = H\mathbf{f}$, where $\mathbf{f}$ is an input signal and $\widetilde{\mathbf{f}}$ is an output signal. Filter H is a renormalized Laplacian matrices and can be decomposed into $H = \mathbf{U}h(\Lambda)\mathbf{U}^{-1}$, where h is a *frequency response function* on a diagonal matrix of eigenvalues Λ. Using Eqs. (15) and (16) with the graph convolutional filter equation, we can decompose $\widetilde{\mathbf{f}} = H\mathbf{f}$ as:

$$\widetilde{\mathbf{f}} = H\mathbf{f} = \mathbf{U}h(\Lambda)(\mathbf{U}^{-1}\mathbf{f}) = \mathbf{U}h(\Lambda)\hat{\mathbf{f}} = \sum_i h(\lambda_i)\hat{f}_i u_i \tag{17}$$

In Eq. (17), the $h(\lambda_i)$ acts as a *low-pass filter*, keeping the low-frequency signals and scaling the high-frequency ones. Hence, we need to use underlying data relation based on structures of graphs to produce proper graph filters which can filter input signals and generate smooth signals for downstream tasks.

Table 3 Performance on the first fold of Cora dataset

Methods	Pruning ratio									
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
SGCN-GCN(%)	**78.59**	**78.23**	**78.60**	**78.23**	**76.75**	**76.38**	**74.17**	**73.06**	**72.33**	**69.01**
RP-GCN(%)	78.59	77.69	76.92	76.55	75.24	74.17	72.66	71.22	68.87	66.97

6.1 Low-pass filters in SGCN-GCN

Recent studies [34] have shown that the normalized graph Laplacian $\widetilde{L} = I_N - \hat{A}$ and filter $\hat{A}$ in GCN can be represented by

$$\hat{A} = I_N - \widetilde{L} = \mathbf{U}(I_N - \widetilde{\Lambda})\mathbf{U}^{-1}. \tag{18}$$

In Eq. (17), the frequency response function $h(\widetilde{\lambda_i})$ is equal to $h(\widetilde{\lambda_i}) = 1 - \widetilde{\lambda_i}$. Combining Eq. (17) with Eq. (18), we can easily derive that the GCN frequency response function is

$$h^2(\widetilde{\lambda_i}) = (1 - \widetilde{\lambda_i})^2. \tag{19}$$

Equation (19) is a quadratic function, where the function is minimum at $\lambda_i = 1$. For maintaining a smooth signal, the best function performance is in range [0, 1], a low-pass area. SGCN-GCN has similar forward model to GCN but obtains $\hat{A} = \widetilde{D_p}^{-\frac{1}{2}} \widetilde{A_p} \widetilde{D_p}^{-\frac{1}{2}}$ and $\widetilde{A_p} = A_p + I_N$ and $\widetilde{D} = \sum_j \widetilde{A_p}_{ij}$, where A_p is a pruned adjacency matrix from SGCN. Hence, a range of eigenvalues $\widetilde{\lambda_i}$ from a Laplacian matrix $\widetilde{L}$ can affect the range of values in this frequency response function. Both of SGCN and RP can provide pruned adjacency matrices as inputs to GCN. For explaining the difference effects of SGCN and RP on the performance of GCN, we use the experimental results of the first fold of Cora dataset to display distribution of eigenvalues and analyze the performance of the frequency response functions.

Here, we note GCN uses subgraphs from SGCN as SGCN-GCN while GCN applies subgraphs from RP as RP-GCN. Table 3 illustrates that the SGCN-GCN performance improves by 2% on average when compared to that of RP-GCN. For further analyzing the frequency response functions and distributions of eigenvalues in filters, we look at the SGCN-GCN and RP-GCN results for pruning ratios of 50% and 90% on the dataset. Figure 7 illustrates the distribution of eigenvalues. Figure 7c, d shows that SGCN produces more eigenvalues in the low-pass area [0, 1] in GCN, and the number of eigenvalues larger than 1 produced by RP are more than those made by SGCN. Also, the range of eigenvalues from filters of SGCN-GCN are smaller than that from RP-GCN. Therefore, in GCN, SGCN shrinks the range of eigenvalues into low-pass areas better than RP does. These effects are applied to frequency response functions, such that SGCN can avoid amplifying eigenvalues and reduce more noise compared to RP in GCN. In next section, we demonstrate how to quantitatively measure strength of the low-pass filter such that we can further explain the effect of the low-pass filter.

6.2 Strength of low-pass filters

Generally, ℓ_2-norm is a natural way to measure magnitude of vectors. Here, we define a way to measure the strength of low-pass filters by using a proportion of square of ℓ_2-norm of signals. A signal can be preserved by Eq. (15) and Eq. (16). In SGCN-GCN, signals are transferred to spectral coefficients by Eq. (15) and the spectral coefficients are transferred back to signals in Eq. (16) scaled by the frequency response function $h()$. That means information of signals can be filtered by filters. Therefore, we propose the following theorem to measure the strength of filters in SGCN-GCN and RP-GCN.

Theorem 1 *Let* $\mathbf{f}$ *be a signal from graph* G*, and* $\mathbf{U} = [u_0, \ldots, u_{n-1}]$ *be an orthogonal matrix of eigenvectors of the Laplacian matrix of* G*. The strength of a filter can be evaluated by* $S = \frac{\|h(\Lambda)\hat{\mathbf{f}}\|_2^2}{\|\hat{\mathbf{f}}\|_2^2}$*, where* $\hat{\mathbf{f}}$ *denotes spectral coefficients,* $h()$ *is a frequency response function, and* Λ *is a matrix of eigenvalues. An estimation of the strength of a filter is equal to* $\mathbb{E}(h(\lambda_i)^2)$*, where* $i = [0, ..n-1]$

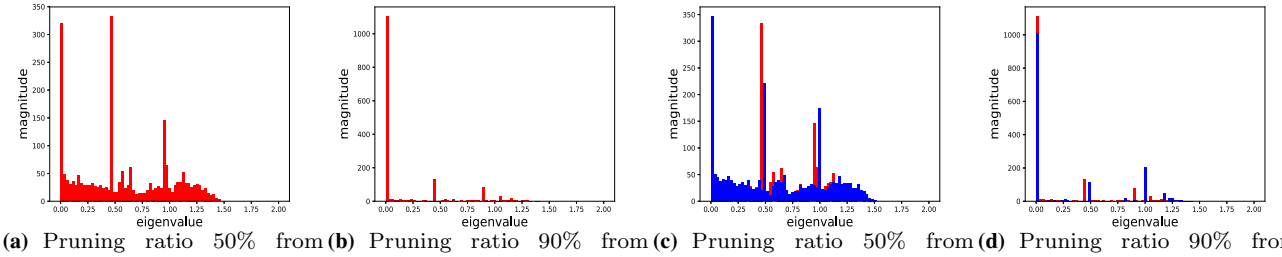
Proof Based on Eq. (15), Eq. (16) and Eq. (17) with orthogonal matrix $\mathbf{U}$ and output signals $\mathbf{f}'$, we have

$$\begin{aligned} S &= \frac{\|\mathbf{f}'\|_2^2}{\|\mathbf{f}\|_2^2} = \frac{\|H\mathbf{f}\|_2^2}{\|\mathbf{U}\mathbf{U}^{-1}\mathbf{f}\|_2^2} = \frac{\|\mathbf{U}h(\Lambda)\mathbf{U}^{-1}\mathbf{f}\|_2^2}{\|\mathbf{U}\mathbf{U}^{-1}\mathbf{f}\|_2^2} \\ &= \frac{\|\mathbf{U}h(\Lambda)\hat{\mathbf{f}}\|_2^2}{\|\mathbf{U}\hat{\mathbf{f}}\|_2^2} = \frac{\|h(\Lambda)\hat{\mathbf{f}}\|_2^2}{\|\hat{\mathbf{f}}\|_2^2}. \end{aligned} \tag{20}$$

Also, we can decompose Eq. (20) as follows:

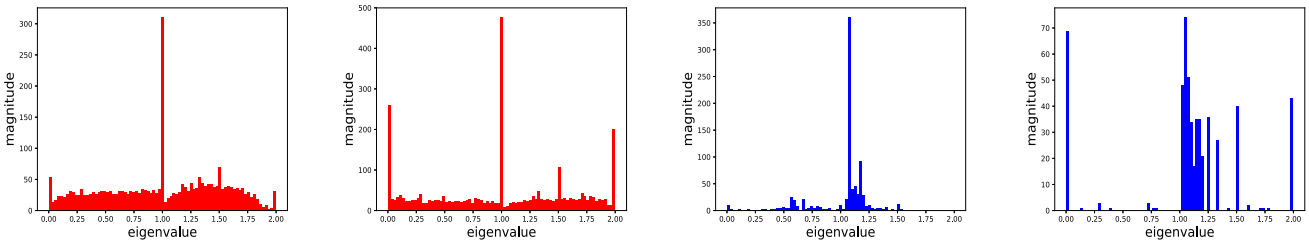
$$\frac{\|h(\Lambda)\hat{\mathbf{f}}\|_2^2}{\|\hat{\mathbf{f}}\|_2^2} = \frac{\sum_i \left(h(\lambda_i)\hat{f}_i\right)^2}{\sum_i \hat{f}_i^2}. \tag{21}$$

In SGCN-GCN and RP-GCN, we know that $h(\lambda_i) = (1 - \lambda_i)^2$. Therefore, the boundary of $\frac{\|\mathbf{f}'\|_2^2}{\|\mathbf{f}\|_2^2}$ is within [0, 1] based on Eq. (21). Considering the minimization $min(\sum_i (h(\lambda_i)^2 - \delta)\hat{f}_i^2)$, we can easily derive the solution as $\delta = \mathbb{E}(h(\lambda_i)^2)$. Hence Eq. (21) can be approximated by



(a) Pruning ratio 50% from SGCN **(b)** Pruning ratio 90% from SGCN **(c)** Pruning ratio 50% from SGCN and RP **(d)** Pruning ratio 90% from SGCN and RP

Fig. 7 Distributions of eigenvalues from SGCN and RP on Cora dataset. Figure 7a, b shows results from SGCN. Figure 7c, d aims to compare results from RP with those from SGCN



(a) Cora **(b)** CiteSeer **(c)** Terrorist **(d)** Terrorist Attacks

Fig. 8 Distributions of eigenvalues for Cora, Terrorist Attacks, CiteSeer and Terrorist

Table 4 Strength of filters on the first fold of Cora dataset. Note that weaker filters are highlighted in Table

Methods	Pruning ratio									
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
SGCN-GCN	0.228	**0.259**	**0.298**	**0.344**	**0.459**	**0.451**	**0.574**	**0.629**	**0.752**	**0.860**
RP-GCN	0.228	0.258	0.293	0.335	0.384	0.441	0.511	0.595	0.701	0.832

Best results are highlighted in bold

$\mathbb{E}(h(\lambda_i)^2)$, which captures how the distribution of eigenvalues can affect the performance of low-pass filters in SGCN-GCN and RP-GCN. □

Based on Theorem 1, when ratio S is closer to 1, a filter is weaker. This is because compared to original signals, the magnitude of current signals are hardly scaled by a filter. When the ratio is approaching 0, the strength of a filter is too strong and it almost filters everything. In terms of Eq. (17) for preserving smoothness, when signals are of high frequency, the filter should be stronger, such that it can produce smooth signals for downstream tasks and have reasonable performance in classification. By contrast, we should apply a weaker filter when processing low-frequency signals.

Based on GSP principals, high-frequency signals correspond to large eigenvalues while low-frequency signals match smaller eigenvalues. Therefore, from Fig. 8a, we know that the Cora dataset consists of many low-frequency signals with some high-frequency signals. Hence, a filter on the dataset should be weaker to keep most low-frequency signal and better scale high-frequency ones. Table 4 illustrates the change of strength of filters with variations in pruning ratios in SGCN and RP on the Cora dataset. Compared to with RP for the same pruning ratio, SGCN produces a weaker filter as shown in the table. This is because SGCN can reduce more eigenvalue to low-pass area and $\mathbb{E}(h(\lambda_i)^2)$ (the strength of low-pass filters S) becomes larger, which shrinks the range of eigenvalues and avoids noise from high-frequency components, as shown in Fig. 7a–d. Therefore, the SGCN filter can handle low-frequency signal better than RP filter does. This also explains Fig. 3a, where SGCN performs better than RP for all pruning ratios.

On the Terrorist Attacks dataset, when pruning ratio increases, the performance of SGCN-GCN and RP-GCN improve. The distribution of eigenvalues on this dataset (Fig. 8d) can explain the scenario. Figure 8d shows that the dataset mostly consists of high-frequency components with large eigenvalues over low-pass area and only has limited information in low-pass area. Therefore, in terms of the Eq. (17) and frequency respond function, a weaker filter should be applied to this dataset, such that it reduces negative effects from high-frequency components and keeps the limited information in low-pass area. As pruning ratio increases, the strength of a filter becomes weaker based on

Theorem 1. Hence the performance of SGCN-GCN and RP-GCN becomes better with weaker filters on the dataset.

The distribution of eigenvalues in the CiteSeer dataset (Fig. 8b) is similar to that of the Cora dataset (Fig. 8a). Hence their filters should act similarly, where filters gradually become weaker and performance decreases with the pruning ratio increasing. Since the distribution of eigenvalues on the Terrorist dataset are mainly high-frequency signals with few low-frequency signals, a weaker filter as pruning ratio increases is reasonable for keeping parts of information in low-pass area and reducing noise from high-frequency components. The experimental results in Fig. 3 of our paper validate this hypothesis.

7 Conclusion

When a graph is large or dense, node classification often requires massive storage or is computationally expensive. In this paper, we address this issue by proposing the first neural network architecture that can sparsify graphs for node classification. We propose *Sparsified Graph Convolutional Network* (SGCN), a neural network sparsifier. In SGCN, we formulate sparsification as an optimization problem and provide an ADMM-based solution to solve it. Experimental results on real-world datasets demonstrate that the proposed framework can sparsify graphs and its output (sparsified graphs) can be used as inputs to GCN to obtain classification models that are as accurate as using the whole graphs. Hence, SGCN reduces storage and computational cost with a limited loss in classification accuracy. Moreover, we define a measure to evaluate strength of SGCN and shed light on the reason why SGCN achieves its performance, which is that SGCN shrinks the range of eigenvalues and maintains sufficient filtering strength for various pruning ratios.

Acknowledgements This research was supported in part by the National Science Foundation under awards CAREER IIS-1942929, CAREER CMMI-1750531, and ECCS-1609916.

References

1. Benczúr, A.A., Karger, D.R.: Approximating *s-t* minimum cuts in $\tilde{O}(n^2)$ time. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22–24, 1996, pp. 47–55 (1996)
2. Bhagat, S., Cormode, G., Muthukrishnan, S.: Node classification in social networks. In: Aggarwal, C.C. (eds.) Social Network Data Analytics, pp. 115–148. Springer, Boston (2011). https://doi.org/10.1007/978-1-4419-8462-3_5
3. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends® Mach. Learn. **3**(1), 1–122 (2011)
4. Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, E.R., Mitchell, T.M.: Toward an architecture for never-ending language learning. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI'10, pp. 1306–1313. AAAI Press (2010)
5. Chen, J., Ma, T., Xiao, C.: FastGCN: fast learning with graph convolutional networks via importance sampling. In: International Conference on Learning Representations (2018)
6. Chen, S., Varma, R., Singh, A., Kovacevic, J.: Signal representations on graphs: tools and applications. CoRR (2015). arXiv:1512.05406
7. Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., Hsieh, C.-J.: Cluster-GCN: an efficient algorithm for training deep and large graph convolutional networks (2019)
8. Cho, K., Merrienboer, B.V., Gulcehre, C., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. Comput. Sci. CoRR (2014). arXiv:1406.1078
9. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering . In: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 29. Curran Associates, Inc. (2016). https://proceedings.neurips.cc/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf
10. Feng, Z.: Spectral graph sparsification in nearly-linear time leveraging efficient spectral perturbation analysis. In: Proceedings of the 53rd Annual Design Automation Conference, DAC '16, pp. 57:1–57:6. ACM, New York (2016)
11. Fung, W.S., Hariharan, R., Harvey, N.J., Panigrahi, D.: A general framework for graph sparsification. In: Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, STOC '11, pp. 71–80. ACM, New York (2011)
12. Geng, X., Zhang, H., Bian, J., Chua, T.: Learning image and user features for recommendation in social networks. In: 2015 IEEE International Conference on Computer Vision (ICCV), pp. 4274–4282 (2015)
13. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry . In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 1263–1272. PMLR (2017). http://proceedings.mlr.press/v70/gilmer17a/gilmer17a.pdf
14. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: NIPS (2017)
15. Hong, M., Luo, Z.Q., Razaviyayn, M.: Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. SIAM J. Optim. **26**(1), 337–364 (2016)
16. Karger, D.R.: Random sampling in cut, flow, and network design problems. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23–25 May 1994, Montréal, Québec, Canada, pp. 648–657 (1994)
17. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks . CoRR (2016). arXiv:1609.02907
18. Li, J., Zhang, T., Tian, H., Jin, S., Fardad, M., Zafarani, R.: SGCN: a graph sparsifier based on graph convolutional networks. In: Lauw, H.W., Wong, R.C.W., Ntoulas, A., Lim, E.P., Ng, S.K., Pan, S.J. (eds.) Advances in Knowledge Discovery and Data Mining, pp. 275–287. Springer, Cham (2020)
19. Lindner, G., Staudt, C.L., Hamann, M., Meyerhenke, H., Wagner, D.: Structure-preserving sparsification of social networks. In: Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015, ASONAM '15, pp. 448–454. ACM, New York (2015)
20. Lü, L., Zhou, T.: Link prediction in complex networks: a survey. Physica A Stat. Mech. Appl. **390**(6), 1150–1170 (2011)

21. Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs . In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of the 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 2014–2023. PMLR, New York (2016). http://proceedings.mlr.press/v48/niepert16.pdf
22. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, pp. 701–710. ACM, New York (2014)
23. Rong, Y., Huang, W., Xu, T., Huang, J.: Dropedge: towards deep graph convolutional networks on node classification. In: International Conference on Learning Representations (2020)
24. Satuluri, V., Parthasarathy, S., Ruan, Y.: Local graph sparsification for scalable clustering. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD '11, pp. 721–732. ACM, New York (2011)
25. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. Trans. Neural Netw. **20**(1), 61–80 (2009)
26. Schaeffer, S.E.: Survey: graph clustering. Comput. Sci. Rev. **1**(1), 27–64 (2007)
27. Sen, P., Namata, G.M., Bilgic, M., Getoor, L., Gallagher, B., Eliassi-Rad, T.: Collective classification in network data. AI Mag. **29**(3), 93–106 (2008)
28. Serrano, M.Á., Boguñá, M., Vespignani, A.: Extracting the multiscale backbone of complex weighted networks. Proc. Natl. Acad. Sci. U. S. A. **106**(16), 6483–8 (2009)
29. Shuman, D.I., Narang, S.K., Frossard, P., Ortega, A., Vandergheynst, P.: Signal processing on graphs: extending high-dimensional data analysis to networks and other irregular data domains . CoRR (2012). arXiv:1211.0053
30. Spielman, D.A., Srivastava, N.: Graph sparsification by effective resistances. In: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing. STOC'08, Victoria, British Columbia, Canada, pp. 563–568. Association for Computing Machinery, New York (2008). https://doi.org/10.1145/1374376.1374456
31. Spielman, D.A., Teng, S.: Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. SIAM J.Matrix Anal. Appl. **35**(3), 835–885 (2014). https://doi.org/10.1137/090771430
32. Takapoui, R., Moehle, N., Boyd, S., Bemporad, A.: A simple effective heuristic for embedded mixed-integer quadratic programming. In: 2016 American Control Conference (ACC), pp. 5619–5625 (2016). https://doi.org/10.1109/ACC.2016.7526551
33. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: International Conference on Learning Representations (2018)
34. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., Weinberger, K.: Simplifying graph convolutional networks. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol. 97, pp. 6861–6871. PMLR, Long Beach (2019)
35. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. IEEE Trans. Neural Netw. Learn. Syst. **32**(1), 4–24 (2019). https://doi.org/10.1109/TNNLS.2020.2978386
36. Yang, Z., Cohen, W.W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning, vol. 48, ICML'16, pp. 40–48. JMLR.org (2016)
37. Zheng, C., Zong, B., Cheng, W., Song, D., Ni, J., Yu, W., Chen, H., Wang, W.: Robust graph representation learning via neural sparsification. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol. 119, pp. 11458–11468. PMLR (2020)
38. Zhu, X.: Semi-supervised learning with graphs. Ph.D. Thesis, Pittsburgh, PA, USA (2005). AAI3179046